Dodging Non-Uniform I/O Access in Hierarchical Collective Operations for Multicore Clusters

Brice Goglin, Stéphanie Moreaud

▶ **To cite this version:**

Brice Goglin, Stéphanie Moreaud. Dodging Non-Uniform I/O Access in Hierarchical Collective Operations for Multicore Clusters. CASS 2011: The 1st Workshop on Communication Architecture for Scalable Systems, held in conjunction with IPDPS 2011, May 2011, Anchorage, United States. 7p, 10.1109/IPDPS.2011.222 . inria-00566246

HAL Id: inria-00566246

https://inria.hal.science/inria-00566246

Submitted on 16 Feb 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Dodging Non-Uniform I/O Access in Hierarchical Collective Operations for Multicore Clusters

Brice Goglin and Stéphanie Moreaud
INRIA, LaBRI, University of Bordeaux – Talence, France
{brice.goglin,stephanie.moreaud}@inria.fr

Abstract

The increasing number of cores led to scalability issues in modern servers that were addressed by using non-uniform memory interconnects such as HyperTransport and QPI. These technologies reintroduced Non-Uniform Memory Access (NUMA) architectures. They are also responsible for Non-Uniform Input/Output Access (NUIOA), as I/O devices may be directly connected to a single processor, thus getting faster access to some cores and memory banks than to the others. In this paper, we propose to adapt MPI collective operations to NUIOA constraints. These operations are now often based on the combination of multiple strategies depending on the underlying cluster topology, with local leader processes being used as intermediate. Our strategy focuses on electing these leaders according to the locality of processes and network interfaces so as to give them privileged network access. We validate our approach on a hierarchical Broadcast operation which brings up to 25% throughput improvement between 64 processes.

1 Introduction

The increasing number of cores in computing nodes raised the need to remove the centralized memory bus bottleneck. Modern scalable systems rely on memory interconnects that distribute banks across the machine. These architectures are called NUMA (*Non-Uniform Memory Access*) as each processor gets high-performance access to its local memory and usually slower access to the remaining machine memory. Affinities between tasks and data was already seen as a significant scheduling criteria on modern machines because of caches, it now becomes even more important because of NUMA effects [16].

Besides these effects, these memory interconnects may also be responsible for *Non-Uniform Input/Output Access* (NUIOA) when some I/O devices are closer to some processors and memory banks than to the others. This property introduces a new kind of affinity between processes, data and I/O devices. It may have to be involved in communication strategies to get optimal performance on todays platforms. Indeed, high-performance computing requires optimizations all along the communication path between processes, both inside and outside the nodes. We previously demonstrated a significant impact on point-to-point performance [12] and we improved multirail communication strategies accordingly [13].

We propose to take NUIOA constraints into account in the implementation of collective operations whose performance and scalability are key components of MPI communication libraries. One common way to improve them is to adapt them to the underlying cluster and node topology. Intra-node and inter-node transfers are implemented separately and may be combined in a hierarchical manner according to locality. Our idea consists in generalizing this problem by looking at I/O device locality.

The remaining of the paper is organized as follows. Section 2 presents current NUIOA platforms and their behavior. Our proposal and implementation of NUIOA-aware collective operations is described in Section 3. The performance

of the corresponding *Broadcast* and *Gather* operations is then evaluated in Section 4 while Section 5 discusses our approach and compares it to other approaches.

2 Non Uniform Input/Output Access

We summarize in this section the existing candidate NUIOA platforms and then detail their actual performance behavior.

2.1 NUMA and NUIOA Architectures

NUMA architectures started spreading into high-performance computing clusters in 2003 when AMD introduced the OPTERON processor and the HYPERTRANSPORT memory interconnect. Since then, the multiprocessor AMD platforms exhibited NUMA and NUIOA effects because each memory bank or I/O chipset is directly connected to a single socket [9]. Local cores and devices get privileged data transfer to this local memory bank. Figures 1(a) and 1(b) present the architecture of modern AMD platforms. Although latest processors may support up to four HYPERTRANSPORT links, each I/O chipset is still only connected to a single socket.

INTEL NUMA platforms were limited to the ITANIUM processors until the introduction of the *Nehalem* architecture in 2008. Thanks to the QPI memory interconnect, all modern INTEL servers are now NUMA [1]. They may also exhibit NUIOA effects depending on how the two QPI links of the I/O hubs are connected. In the common case, two processors and a single I/O hub are fully interconnected (as depicted in Figure 1(c)), making the machine NUMA but not NUIOA. However, larger servers may have more I/O hubs and/or sockets, making the platform NUIOA. Figures 1(d) and 1(e) illustrate such architectures.

The advent of the *Sandy-Bridge* architecture in 2011 is expected to bring NUIOA into all INTEL platforms. Indeed a I/O hub will be integrated inside the socket. It will thus have naturally privileged access to the local cores and memory.

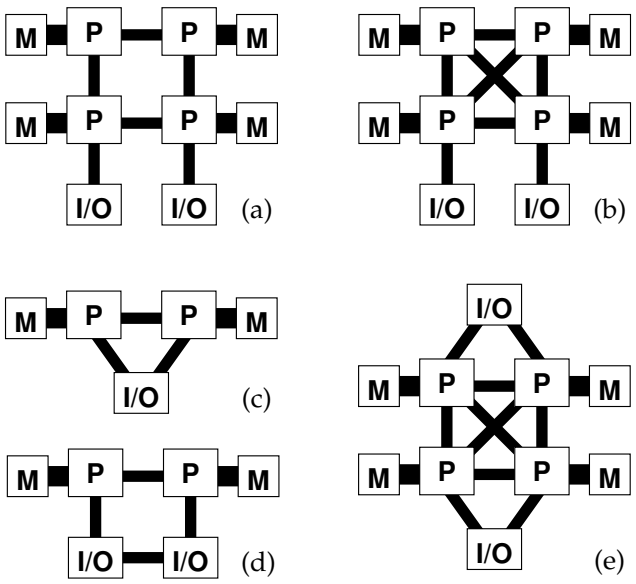

Figure 1. Interconnection of processors (P), memory (M) and I/O chipsets in some modern platforms: (a) four AMD Istanbul processors; (b) four AMD Magny-Cours processors; (c) and (d) two Intel Westmere-EP processors; (e) four Intel Nehalem-EX processors.

2.2 I/O Affinities

NUIOA platforms offer privileged access to I/O devices to some cores due to their lower physical distance. This constraint has been known to impact I/O performance for a long time. For instance, all high-speed networking microbenchmarks are manually bound to a core near the network interface (NIC) so as to achieve optimal performance. We studied this feature on old OPTERON servers and noticed significant performance variations depending on the underlying network technology (up to 40% throughput degradation for some multirail transfers [12]). NUIOA is sometimes also referred to as *Non-Uniform Network Access* (NUNA) but is actually not specific to network devices. Indeed, we observed DMA throughput decrease by up to 42% when accessing a NVIDIA GPU from the distant NUMA node in a machine depicted in Figure 1(d).

There are actually two ways to work around NUIOA constraints. One is to change the process

placement to have communication-intensive tasks near the NICs. We demonstrated the ability to automatically bind tasks near the NIC by gathering I/O affinity informations from the operating system [12]. However, detecting the communication-intensiveness may not be easy and some applications have uniform communication patterns anyway. Moreover, this approach may conflict with other placement policies that may be decided based on affinities between processes [7].

Another approach consists in considering a given process distribution and adapting the communication strategy accordingly. We showed in previous papers that the implementation of multi-rail communication in MPI libraries may be tuned to benefit from I/O affinities [13]. In this paper, we propose to look at I/O affinities in the context of MPI collective operations.

3 NUIOA-aware Collective Operations

Collective operations are widely used in parallel applications, either for synchronization purpose or for distribution/gathering of data among the processes. Implementing such operations on modern clusters with hundreds of nodes and dozens of cores per nodes obviously raised scalability problems due to the latency requirements or to the communication-intensive patterns they may involve. This section first describes how collective operations are implemented in modern MPI layers. We then propose a way to take I/O affinities into account in these implementations.

3.1 Collective Operations on Many-core Clusters

The increasing number of nodes in clusters led to a rethink of the implementation of collective operations. Indeed, a naive *Broadcast* operation (root process sending to all processes independently) could likely hit scalability limits considering the contention on its outgoing link. Many optimizations such as binary-tree algorithms are thus involved in algorithms [15].

These implementations provide a better ability to map the communication pattern onto the underlying hardware data transfer protocols. On modern manycore clusters, this aim is supported by improvement on the *intra-node* side of collective operations which benefits from the shared-memory between local processes [4, 11]. Intra-node optimizations are now often combined with inter-node communication within hierarchical algorithms. For an *Alltoall* operation, this idea may be implemented through an intra-node *Alltoall* on each node, followed by inter-node *Alltoall* between all groups of corresponding local ranks [10].

3.2 NUIOA-aware Leadership

Hierarchical collective implementations are usually based on the election of leader processes responsible for representing multiple local processes. Our proposal is to **modify this leader election according to I/O affinities**. As the leader is responsible for performing operations on the inter-node network, its privileged access to the network interface should improve the overall performance. Moreover, as depicted on Figure 2, the location of the leader may reduce congestion on the memory interconnect.

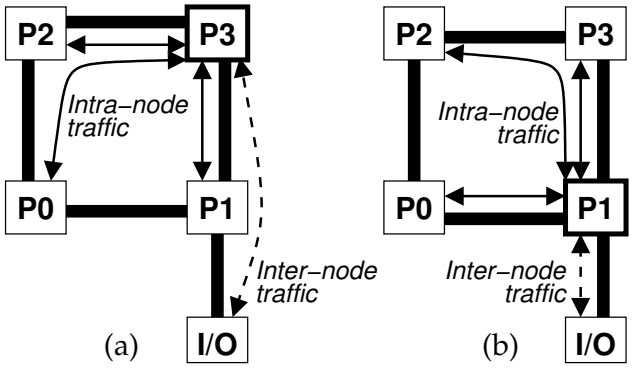

Figure 2. Traffic on memory interconnect depending on the leader. (a) The leader (P3) is far from the network interface. The memory link between P1 and P3 may be congested because intra-node and inter-node traffics conflict. (b) The leader (P1) is near the interface. The inter-node traffic does not overload the internal memory interconnect.

We previously demonstrated that NUIOA ef-

fects mostly matter on the target side [12]. For instance, it means that the placement is more important for the receiver process than for the sender in case of a MPI two-sided communication, or for the initiator than for the remote process in case of a RDMA *Get* operation. Also only large messages (hundreds of kilobytes) are subject to NUIOA since only large bandwidths are likely to suffer from distances (the impact on latency is in the order of one hundred nanoseconds). We verified that these results are still valid on todays INTEL and AMD platforms (see Section 4.2). It means that our NUIOA-aware election of local leaders should actually primarily focus on processes that receive some data from the network: either the final destinations of inter-node transfers, or the intermediate leaders in multi-step algorithms.

3.3 Implementation

We implemented our idea in the OPEN MPI 1.5 library [3] which offers several collective components [14]. The default collective component, called *Tuned* [2], switches between different algorithms (pipeline, binary-tree, *etc.*) depending on the message size and segmentation, and on the number of processes and their location. Our work however focuses on another component, called *Hierarch*, which implements several collective operations in a hierarchical manner. While being less optimized than *Tuned*, the relative simplicity of the hierarchical component offers more room for study and improvement, especially in the area of locality in scalable systems.

The *Hierarch* component combines multiple collective components by splitting collective operations into several steps that usually match different hardware hierarchy levels. It especially offers an easy way to combine shared-memory collectives inside the nodes and network-based collectives between nodes. For the *Broadcast* operation, the algorithm consists in the root process sending the data to one process on each node, then all of them broadcast the message to the other processes on their node.

By default, these intermediate processes on each node (the local leaders) are elected so that their local rank is the same than the root local rank. As depicted on Figure 3, we broke this rule so that leaders are near a network interface. This is implemented by using the HWLOC (*Hardware Locality*) library which is able to report which NUMA node is closer to some PCI devices. Combined with HWLOC ability to bind processes, it enables the full knowledge of process affinities with respect to the INFINIBAND cards used in our testbed.

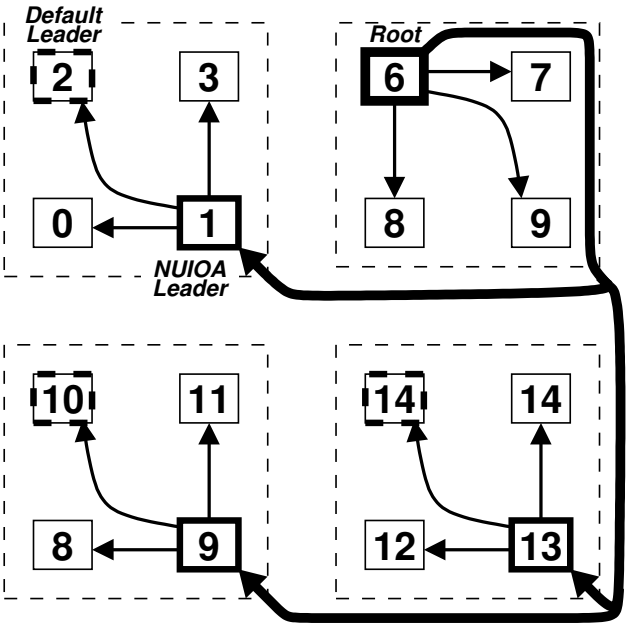

Figure 3. Communication scheme during a NUIOA Broadcast between 16 processes on 4 nodes. The root (process #6) uses processes #1, #9 and #13 as leaders on other nodes since they are near their network interface. The default (non-NUIOA) algorithm would use processes #2, #10 and #12 as leaders instead.

All inter-node communication occur between the root and a leader, or even between two leaders if the inter-leader collective is also hierarchical. Thanks to our idea, only the root process may access the external network without being close to its network interface. We now look at the suitability of our NUIOA-aware leader election strategy considering various collective operations.

3.3.1 One-to-all

In the case of **One-to-all** collectives such as *Broadcast*, the root process sends data to other processes without receiving significant amounts of payload. As explained earlier, NUIOA effects mostly matter for receiving on current platforms. The non-NUIOA placement of the root should thus be negligible. Meanwhile, all receiving processes (all other leaders) are properly chosen near a network interface so as to avoid NUIOA issues. This case matches perfectly our idea and we expect an immediate performance improvement.

3.3.2 All-to-one

All-to-one operations such as *Gather* are harder to optimize since the root process will receive messages while possibly being far from its local interface. Modifying the binding of the root process is not an acceptable solution since the binding may have been chosen for other reasons. Also multiple collectives may use different root processes.

We feel that NUIOA constraints are not applicable to this case without breaking the hierarchical algorithm. One possible solution would be add an intermediate process near the NIC and have it the incoming traffic to the root, but it is not clear that this would improve the overall performance.

3.3.3 All-to-all

All-to-all collective operations such as *Allgather* have no explicit root but may actually be implemented with local leaders that gather, exchange and scatter the data of all local processes. All these local leaders may easily be elected near the network interface, and thus avoiding all NUIOA issues.

4 Performance Evaluation

4.1 Experimentation Platform

The experimentation platform consists of eight quad-socket hosts with dual-core OPTERON 8218 processors (2.6 GHz). As depicted by Figure 1(a), each host contains four NUMA nodes, two of them being also connected to their own I/O bus. The cluster is interconnected with MELLANOX MT25418 CONNECT-X DDR INFINIBAND cards. These cards are plugged on a PCIe 8x slot behind the second I/O bus of each host, hence near NUMA node #1 (as shown on Figure 2).

These hosts run the *Intel MPI Benchmarks* (IMB [5]) on top of our modified OPEN MPI 1.5 implementation. Only the *Broadcast* operation was modified since the hierarchical component does not support all collectives yet. We also present the impact of NUIOA binding of processes on the performance of *Gather* to demonstrate that hierarchical *All-to-one* operations are not good candidates for this work (as explained in Section 3.3.2).

4.2 Point-to-point Microbenchmarks

We first look at the impact of NUIOA effects on point-to-point operations so as to better understand the following collective performance evaluation. Figure 4 presents the throughput of a MPI ping-pong measured with the *Intel MPI Benchmarks* between 2 processes on different nodes depending on the process placement with respect to the INFINIBAND interface. It shows that the throughput of large messages (starting at 32 kB) may be increased by 25-30% if the processes are running on a core near the local interface. For smaller messages, the improvement is about 5% (about 180 ns latency difference).

Figure 4 also shows the throughput of a IMB Broadcast between these processes. The *Basic* OPEN MPI component implements this operation as a single message from the root to the second process. As expected, its performance and the impact of locality is very similar to the one of the Pingpong.

However, this linear broadcast algorithm obviously does not scale well with the number of nodes and cores. We thus also present the performance of the default component (*Tuned*) which uses advanced and more scalable strategies such as pipelines and binary-trees as explained in Section 3.3. Its performance is far for optimal in our case because it already pipelines messages when

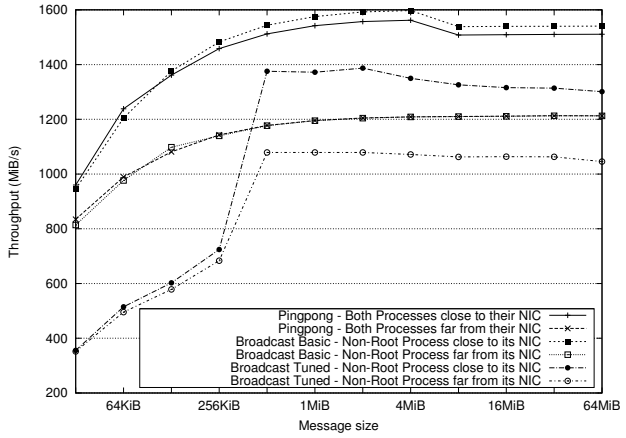

Figure 4. IMB Pingpong and Broadcast throughput between two processes on different machines depending on their placement.

only two processes are involved, hence reducing the overall throughput. However, the relative impact of locality is similar to those of Pingpong and *Basic* broadcast.

These unidirectional tests let us verify that only the locality of the target matters. Indeed, modifying the placement of the root (sender) process does not modify the observed throughput. This asymmetric behavior seems to be caused by a saturation of the HYPERTRANSPORT memory interconnect when data is transfered from an I/O chipset to a socket. We actually observed a possibly similar phenomenon on INTEL-based platforms such as Figure 1(d). On the other hand our AMD platform exhibits much smaller NUIOA effects when using another networking technology (MYRICOM MYRI-10G). One explanation for this asymmetric behavior could be that the INFINIBAND NIC generates many small packets when writing in the host memory by DMA (the number of request and response packets that may be in flight at the same time on the memory interconnect is limited in hardware).

4.3 Inter-node NUIOA Broadcast Performance

We now study the performance of actual collective operations between our eight hosts. Figure 5 presents the per-process throughput of a Broadcast between one process on each node. This test does not involve the hierarchical component as a single process is running on each node. Only a inter-node collective operation is performed. We compare the overall performance depending on the binding of all processes. Again, this graph confirms that only the location of the target processes (non-root) is important, bringing up to 50% better throughput. Carefully binding the root process near its INFINIBAND interface does not help performance.

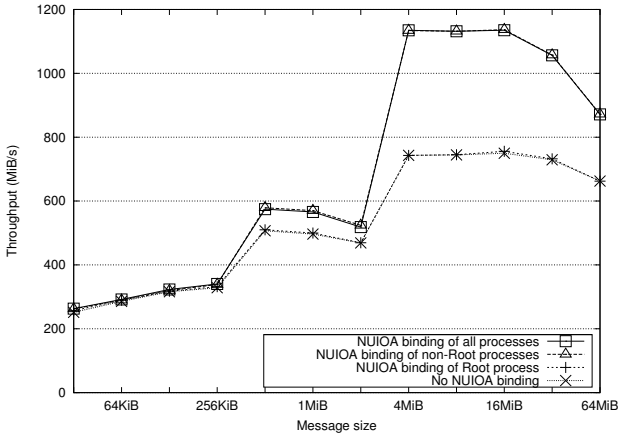

Figure 5. Performance of a Broadcast between 8 processes (one per node) depending on the process binding.

4.4 Inter-node NUIOA Gather Performance

The previous section confirmed that *One-to-all* collective operations match the behavior of NUIOA effects. We now look at a *All-to-one* operation to verify that our NUIOA leader election cannot help as expected from Section 3.3.2.

Figure 6 presents the performance of a Gather operation between one process per node depending on their binding. It confirms that only the location of the root process matters: the aggregate throughput is improved by 47% when placing the root process near the INFINIBAND card. Indeed, the root process is the actual receiver of all data transfer during a Gather.

Improving the locality of a *All-to-one* operation requires to change the binding of the root process.

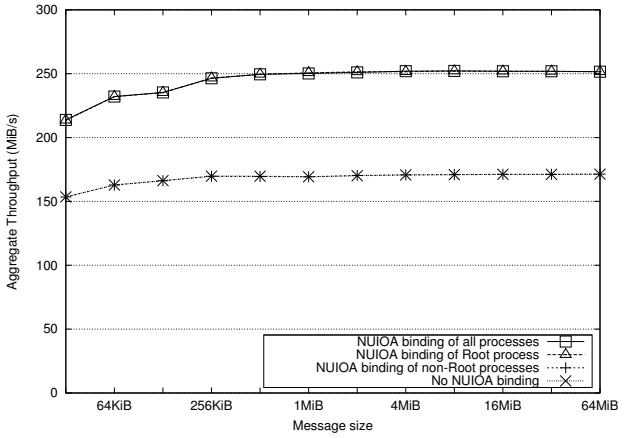

Figure 6. Performance of a Gather between 8 processes (one per node) depending on the process binding.

However, this is may be hard if all operations during the execution do not always have the same root.

We will therefore focus on *One-to-all* operations in the remaining of this paper.

4.5 Hierarchical NUIOA Broadcast Performance

We now use one process per core. Local leaders are now playing their intermediate role in the hierarchical algorithm. Figure 7 presents the per-process Broadcast throughput depending on the leader and on the collective implementation. Both variants of the hierarchical component show that our NUIOA-aware election of local leaders brings a significant performance improvement (from 10% to 25%) as expected.

The standard variant (labelled as *hierarchical*) uses point-to-point operations for the inter-leader broadcast while the collective shared-memory component is used inside each node. The *non-blocking pipelined* (labelled as *nbp*) variant uses non-blocking point-to-point operations for both inter-node and intra-node communication, and splits messages in 256kB chunks so as to pipeline the steps of the hierarchical algorithm. Although this article does not focus on comparing various collective implementations, we want to emphasize the fact that we observed performance improvement thanks to our NUIOA leaders with many different models and tunings of the hierarchical component.

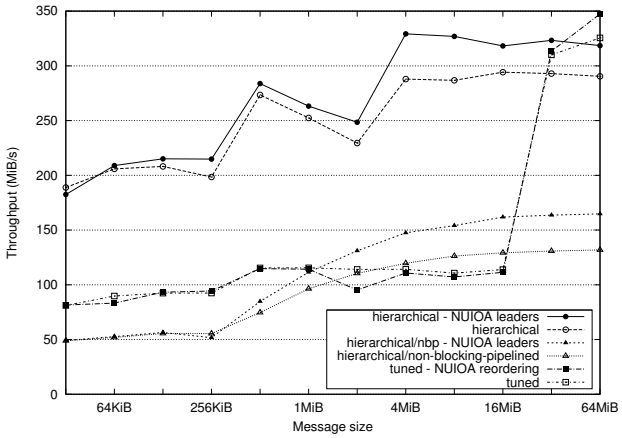

Figure 7. Performance of a Broadcast between 64 processes (one per core, eight per node) depending on the algorithm.

We actually even observed benefits from NUIOA-awareness on the default collective component (*Tuned* [2]) which is not hierarchical. It builds a linear chain of processes that propagate the message in a pipelined manner (the chunk size depends on the message size). Given the default process ordering, it means that the message first goes from the root to the next local processes, then to the first process on the next node, then to other processes on this node, *etc.* We were able to reorder this chain so that each node first receives the message in a process near the INFINIBAND interface. Figure 7 shows that the *Tuned* component is not actually well tuned for our machine, but our reordering improves performance when *Tuned* achieves interesting throughput.

4.6 NUIOA Broadcast Scalability

We now look deeper at the impact of our NUIOA leader election depending on the number of nodes and processes per node. Figure 8 presents the aggregate Broadcast throughput when increasing the number of processes on each of the 8 nodes. As expected, the relative performance improvement decreases from 50% to 10%

because the overhead of the intra-node operation increases while the inter-node operation remains the same.

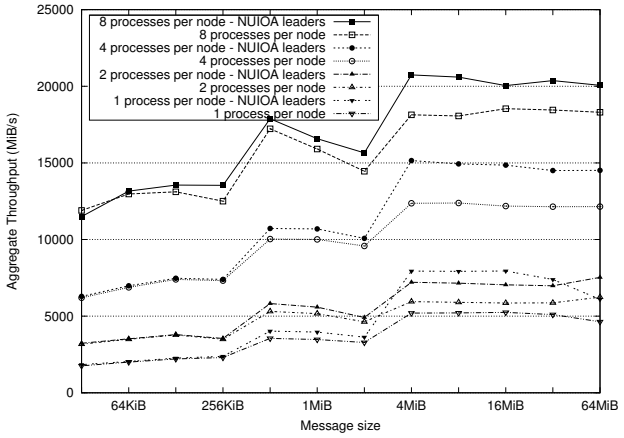

Figure 8. Aggregate throughput of the hierarchical Broadcast when using 1, 2, 4 and 8 processes on each of the 8 nodes.

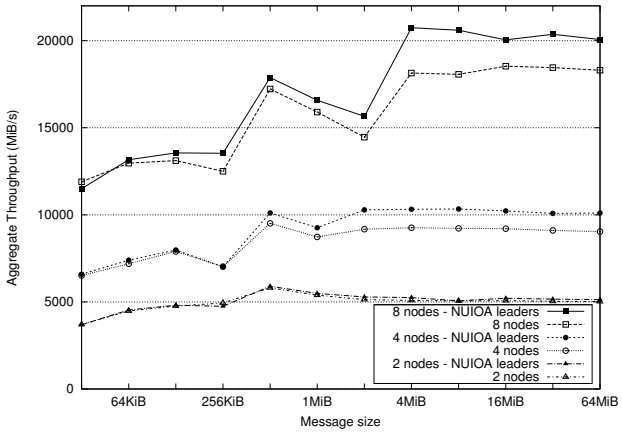

Figure 9. Aggregate throughput of the hierarchical Broadcast when using 2, 4 and 8 nodes with 8 processes each.

Figure 9 now increases the number of nodes while using one process per core on each of them. The impact of our NUIOA leader election increases from 2% with 2 nodes up to 10% when using 8 nodes. Indeed, the intra-node part of operation remains the same while the cost of the inter-node operation increases with the number of nodes.

These results show that our idea makes sense for modern clusters with many nodes containing many cores: The NUIOA leader election always improves the inter-node part of the collective operation without modifying the intra-node part. It improves the overall collective performance significantly as long the number of nodes is not dramatically lower than the number of cores per node.

5 Discussion and Related Works

The election of local leaders based on I/O device locality raises the question of whether our NUIOA leaders may be overloaded. Indeed, the default algorithm distributes the leaders load among all local processes depending on the root process. Fortunately there are multiple cores near each network interface (each socket contains at least 4 cores in modern servers). We are only distributing the load among a socket instead of among the entire server (usually 2 or 4 sockets). Moreover, the current MPI standard only offers blocking collective operations, so the other non-leader cores usually cannot do any useful computation while the leader is working on the collective.

The NUIOA leader election requires the full knowledge of process and I/O location. The former is usually determined by the MPI process manager. It will therefore also have to gather I/O affinity information so as to know which process to elect. For now, our implementation assumes that all nodes have the same number of cores and the same I/O locality, and that local ranks are distributed on each node in the same way. The proper implementation will be straightforward once HWLOC support will be available in OPEN MPI (planned in release 1.5.2).

Another way to work around NUIOA effects in collective operations would be to bind the root process near the network interfaces. Unfortunately, multiple collectives often use different roots. Some communication-pattern-aware placement policies have been proposed [7], they could

be extended to place communication-intensive processes (or processes that are often the root of collective operations) near a network interface. We demonstrated that automatic placement of communicating tasks may significantly help performance [12]. Moreover scheduling and placing processes depending on interrupt affinity and processor topology can reduce the CPU overhead in the context of TCP/IP [6]. We feel that these ideas could be combined in a more general-purpose process placement policy for MPI applications.

Our approach cannot however be applied to some hierarchical algorithms that use multiple leaders per node for inter-node communication [10, 8]. This approach has the advantage of distributing the leaders load across multiple cores. However we feel that it may hit some scalability issues when several dozens of cores will be available in each node due to the contention when accessing the NIC. As explained earlier, using all the cores that are near the interface might be a good compromise because it distributes the load among several cores while maintaining I/O affinity.

6 Conclusion and Future Works

As the number of cores per node increases, the scalability of the servers comes from NUMA architectures which now also often exhibit non-uniform I/O accesses. The locality of the communicating tasks with respect to the network interface has a significant impact on the communication performance. This locality should be taken into account when placing processes and/or implementing high performance communication layers.

We presented in this article a study of the impact of NUIOA architectures on collective operations in parallel applications. As the critical aspect of NUIOA effects on our platform is to have receiving processes near the local interface, we introduced a NUIOA-aware hierarchical Broadcast algorithm that modifies the leaders election to privilege their network access without changing the process binding. We also gave insights on how to modify other collective operations. Performance evaluation shows up to 50% per-process Broadcast throughput improvement thanks to our modification when using one process per node and eight nodes, and up to 25% with eight processes per node. The absolute cost of the operation is reduced, leading to a significant improvement as soon as the number of nodes increases.

We are now looking at implementing similar modifications to other collective operations in the OPEN MPI hierarchical component as well as in other multi-leader based implementations [8]. We also plan to look at combining this work with our earlier results on multirail communication [13]. Indeed servers with multiple NICs may have different cores near each NIC, leading to even more opportunity for taking I/O locality into account in collective operations.

References

[1] I. Corp. An Introduction to the Intel QuicPath Interconnect, Jan. 2009.

[2] G. Fagg, G. Bosilca, J. Pjeivac-Grbovi, T. Angskun, and J. Dongarra. Tuned: An Open MPI Collective Communications Component. In P. Kacsuk, T. Fahringer, and Z. Nmeth, editors, *Distributed and Parallel Systems*, pages 65–72. Springer, 2007.

[3] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, Sept. 2004.

[4] R. L. Graham and G. Shipman. MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives. In *Proceedings of the 15th European PVM/MPI Users' Group Meeting, Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 5208 of *Lecture Notes In Computer Science*, pages 130–140, Dublin, Ireland, Sept. 2008. Springer-Verlag.

[5] Intel MPI Benchmarks. `http://software.intel.com/en-us/articles/intel-mpi-benchmarks/`.

[6] H.-C. Jang and H.-W. Jin. MiAMI: Multi-core Aware Processor Affinity for TCP/IP over Multiple Network Interfaces. In *Proceedings of the 17th Annual Symposium on High-Performance*

Interconnects (HotI'09), pages 73–82, New York, USA, Aug. 2009.

[7] E. Jeannot and G. Mercier. Near-optimal placement of MPI processes on hierarchical NUMA architecture. In *Proceedings of the 16th International Euro-Par Conference, Lecture Notes in Computer Science*, volume 6272 of *Lecture Notes in Computer Science*, Ischia, Italy, Aug. 2010. Springer.

[8] K. Kandalla, H. Subramoni, G. Santhanaraman, M. Koop, and D. K. Panda. Designing Multi-Leader-Based Allgather Algorithms for Multi-Core Clusters. In *CAC 2009: The 9th Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2009*, Roma, Italy, May 2009. IEEE Computer Society Press.

[9] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The AMD Opteron Processor for Multiprocessor Servers. *IEEE Micro*, 23(2):66–76, Mar. 2003.

[10] R. Kumar, A. Mamidala, and D. K. Panda. Scaling Alltoall Collective on Multi-core Systems. In *Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS '08*, Miami, USA, Apr. 2008. IEEE Computer Society Press.

[11] A. R. Mamidala, R. Kumar, D. De, and D. K. Panda. MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics. In *Proceedings of the Int'l Symposium on Cluster Computing and the Grid (CCGrid)*, Lyon, France, May 2008. IEEE Computer Society Press.

[12] S. Moreaud and B. Goglin. Impact of NUMA Effects on High-Speed Networking with Multi-Opteron Machines. In *The 19th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2007)*, Cambridge, USA, Nov. 2007. ACTA Press.

[13] S. Moreaud, B. Goglin, and R. Namyst. Adaptive MPI Multirail Tuning for Non-Uniform Input/Output Access. In E. G. Rainer Keller and J. Dongarra, editors, *Recent Advances in the Message Passing Interface. The 17th European MPI User's Group Meeting (EuroMPI 2010)*, volume 6305 of *Lecture Notes in Computer Science*, pages 239–248, Stuttgart, Germany, Sept. 2010. Springer-Verlag.

[14] J. M. Squyres and A. Lumsdaine. The component architecture of open MPI: Enabling third-party collective algorithms. In V. Getov and T. Kielmann, editors, *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*, pages 167–185, St. Malo, France, July 2004. Springer.

[15] R. Thakur and W. Gropp. Improving the Performance of Collective Operations in MPICH. In *Proceedings of the 10th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2003), Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 2840 of *Lecture Notes in Computer Science*, pages 257–267, Venice, Italy, Sept. 2003. Springer.

[16] R. Yang, J. Antony, P. P. Janes, and A. P. Rendell. Memory and Thread Placement Effects as a Function of Cache Usage: A Study of the Gaussian Chemistry Code on the SunFire X4600 M2. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks (i-span 2008)*, pages 31–36, 2008.